Amendment to the Claims:

This listing of claims replaces all prior versions, and listings, of claims in the application:

1. (Withdrawn) A method of synchronizing threads in a multiple thread system, comprising:

defining an entity which maintains a count of values which increases the value maintained by the object; and

defining a check operation for said element in which, during the checking operation, a calling thread is suspended, and the check is suspended until the value maintained by the entity has reached or exceeded a given value.

2. (Withdrawn) A method as in claim 1 which said entity is allowed only to increment between allowable values, and not to decrement its value.

3. (Withdrawn) A method as in claim 1 wherein said entity is a counter that is only allowed to include integers.

4. (Withdrawn) A method as in claim 3 wherein an initial value of the counter is zero.

5. (Withdrawn) A method as in claim 1, wherein said entity is a/are flags.

6. (Withdrawn) An apparatus comprising a machine-readable storage medium having executable instructions for managing threads in a multithreaded system, the instructions enabling the machine to:

define an entity which maintains a count of values and which is allowed to increment between allowable values;

determine a request for value of the element from a calling thread; and

establish a check operation for said element in which said calling thread is suspendeduntil the entity reaches a predetermined value.

7. (Withdrawn) An apparatus as in claim 6, wherein said entity is a monotonically increasing counter.

8. (Withdrawn) An apparatus as in claim 6, wherein said entity is a flag.

9. (Withdrawn) A apparatus as in claim 6 wherein said system has a plurality of processors therein, wherein each of said processors is running at least one different ones of said threads.

10. (Withdrawn) A method as in claim 1, further comprising defining an error for an operation that decreases the value maintained by the object to occur concurrently with any check operation on the object.

11. (Withdrawn) A method as in claim 1, wherein the value maintained by the object is a numeric value and the increment operation increases the value by a numeric amount.

12. (Withdrawn) A method as in claim 1, wherein the value maintained by the object is a Boolean value or a binary value and the increment operation is a "set" operation that changes the value from one state to the other state.

13. (Withdrawn) A method as in claim 2, wherein the value maintained by the object is a Boolean value or a binary value and the increment operation is a "set" operation that changes the value from one state to the other state.

14. (Withdrawn) A method as in claim 12, further comprising establishing an error for an increment operation on the object to occur more than once.

15. (Withdrawn) A method of defining program code, comprising:

determining different parts of a program which can be executed either sequentially, or in multithreaded parallel by different threads, and which has equivalent results when executed in said sequential or multithreaded parallel; and

defining said different parts as being multithreadable.

16. (Withdrawn) A method as in claim 15 wherein said determining is based on a set of conditions that are sufficient to ensure the equivalence of sequential and multithreaded execution of a program construct.

17. (Withdrawn) A method as in claim 15 wherein said different parts are defined as being multithreadable using an equivalence annotation within the program code.

18. (Withdrawn) A method as in claim 17 wherein said annotation is a pragma.

19. (Withdrawn) A method as in claim 17 wherein said annotation is a code comment.

20. (Withdrawn) A method as in claim 15 further comprising, within said code, multithreaded constructs, in addition to said multithreadable parts.

21. (Withdrawn) A method as in claim 15 wherein said multithreadable parts includes information which, if executed as threads, will include the same result as if executed sequentially.

22. (Withdrawn) A method as in claim 15 wherein said part is a multithreadable block of information.

23. (Withdrawn) A method as in claim 22 wherein said part is a multihreadable FOR loop.

24. (Withdrawn) A method as in claim 15 further comprising synchronizing threads using a monotonically-increasing counter.

25. (Withdrawn) A method as in claim 15 further comprising synchronizing threads using a flag.

26. (Withdrawn) A method as in claim 16, wherein the equivalence annotation includes a new or existing keyword or reserved word in the program.

27. (Withdrawn) A method as in claim 16, wherein the equivalence annotation takes the form of a character formatting in the program, which can be such as boldface, italics, underlining, or other formatting.

28. (Withdrawn) A method as in claim 16, wherein the equivalence annotation takes the form of a special character sequence in the program.

29. (Withdrawn) A method as in claim 16, wherein the equivalence annotation is contained in a file or other entity separate from the program.

30. (Withdrawn) A method as in claim 16, wherein the sequential interpretation of the execution of the block construct is that statements are executed one at a time in their

textual order, and the multithreaded interpretation of the execution of the block construct is that statements of are partitioned among a set of threads and executed concurrently by those threads.

31. (Withdrawn) A method as in claim 16 further comprising using monotonic thread synchronization to synchronize actions among threads.

32. (Withdrawn) A method as in claim 15 wherein:

explicitly multithreaded program constructs are always executed according to a multithreaded interpretation

multithreadable program constructs are either executed according a multithreaded interpretation or executed according to a sequential interpretation; and

sequential or multithreaded execution of multithreadable program constructs is at user selection.

33. (Withdrawn) A method as in claim 32, wherein the sequential or multithreaded execution of multithreadable program constructs is signalled by a pragma in the program.

34. (Withdrawn) A method as in claim 32, wherein the method for selecting sequential or multithreaded execution of multithreadable code constructs is a variable that is dependent of the value of a variable defined in the program or in the environment of the program.

35. (Withdrawn) A method of claim 32 wherein said multiple threaded construct is a block or for loop.

36. (Withdrawn) A method of coding a program, comprising:

defining a first portion of code which must always be executed according to multithreaded semantics, as a multithreaded portion of code;

defining a second portion of code, within the same program as said first portion of code, which may be selectively executed according to either sequential or multithreaded techniques, as a multithreadable code construct; and

allowing a program development system to develop said multithreadable code construct as either a sequential or multithreaded construct.

37. (Withdrawn) A method as in calim 36, wherein said program development system includes a compiler.

38. (Withdrawn) A method as in claim 36 wherein said multithreaded construct defines an operation which has no sequential equivalent.

39. (Withdrawn) A method as in claim 38 wherein said multithreaded construct is control of multiple windows in a graphical system.

40. (Withdrawn) A method as in claim 38 wherein said multithreaded construct is control of different operations of a computer.

41. (Withdrawn) A method as in claim 37 wherein said operation is executed on a multiple processor system, and different parts of said operation are executed on different ones of the processors.

42. (Withdrawn) A method as in claim 37 wherein said multithreadable constructs include a synchronization mechanism.

43. (Withdrawn) A method as in claim 42 wherein said synchronization mechanism is a monotonically increasing counter.

44. (Withdrawn) A method as in claim 43 wherein said synchronization mechanism is a special flag.

45. (Withdrawn) A method of integrating a structured multithreading program development system with a standard program development system, comprising:

detecting program elements which include a specified annotation;

calling a special program development system element which includes a processor that modifies based on the annotation to form a preprocessed file; and

calling the standard program development system to compile the preprocessed file.

46. (Currently Amended) A method of operating a programming language, comprising:

defining equivalence annotations within the programming language which indicate to a program development system of the programming language information about sequential execution of statements written within the programming language and associated with said notations; and

developing the programs as a sequential execution or as a substantially simultaneous execution, based on contents of the equivalence annotations.

47 - 48. (Canceled)

49. (Currently amended) A method of operating a programming language, comprising:

defining equivalence annotations within the programming language which indicate to a program development system of the programming language information about sequential execution of statements written within the programming language and associated with said notations, wherein said ~~multithreaded~~ statements must be executed in a multithreaded manner; and

developing the ~~programs~~ programming language as a sequential execution or as a substantially simultaneous execution based on contents of the equivalence annotations.

50. (Canceled)

51. (Currently Amended) A method of operating a programming language, comprising:

defining equivalence annotations wherein said equivalence annotations are pragmas within the programming language, which indicate to a program development system of the programming language, information about sequential execution of statements written within the programming language and associated with said equivalence annotations; and

developing ~~the~~ programs as a sequential execution or as a substantially simultaneous execution based on contents of the equivalence annotations.

52 - 54. (Canceled)

55. (Currently Amended) A method of operating a program language, comprising:

defining equivalence annotations within ~~the~~ a programming language which indicate to a program development system of the programming language information about sequential execution of statements written within the programming language and associated with said annotations wherein the ~~equivlance~~ equivalence annotations indicate that the statements are multithreadable;

developing ~~the~~ programs as a sequential execution or as a substantially simultaneous execution, based on contents of the equivalence annotations; and

synchronizing access of threads to shared memory using a specially defined synchronization;

synchronization counter;

wherein said synchronization counter is monotonically increasing, cannot be decreased, and prevents thread operation during its check operation.

56. (Canceled)

57. (Currently amended) A method of operating a program language, comprising:

defining equivalence annotations within the program~~ming~~ language which indicate to a program development system of the programming language information about sequential execution of statements written within the programming language and associated with said notations wherein the equivalence annotations indicate that the statements are multithreadable;

developing ~~the~~ programs as a sequential execution or as a substantially simultaneous execution, based on contents of the equivalence annotations; and

synchronizing access of threads to shared memory using a specially defined synchronization;

wherein said synchronization flag is monotonically increasing, cannot be decreased, and prevents thread operation during its check operation.

58. (Currently amended) A method of operating a program language, comprising:

defining equivalence annotations wherein the ~~equivlance~~ equivalence annotations indicate that ~~the~~ statements are multithreadable within the programm~~ing~~ language which indicate to a program development system of the programming language information about sequential execution of said statements written within the programming language and associated with said notations;

developing ~~the~~ programs as a sequential execution or as a substantially simultaneous execution based on contents of the equivalence annotations; and

synchronizing access of threads to shared memory using a specially defined synchronization counter; and

wherein said synchronization counter includes a check operation which suspends a calling thread.

59. (Original) A method as in claim 58 further comprising maintaining a list of suspended threads.

60. (Withdrawn) A method of modifying an existing program development system and environment, comprising:

· detecting which components of a program contain multithreadable program constructs or explicitly multithreaded program constructs;

· transforming the components of the program that contain multithreadable program constructs or explicitly multithreaded program constructs into equivalent multithreaded components in a form that can be directly translated or executed by the existing program development system; and

invoking the existing program development system to translate or execute the transformed components of the program.

61. (Withdrawn) A method as in claim 60, wherein said indicating comprises giving distinctive names to said component.

62. (Withdrawn) A method as in claim 59, wherein the transforming of the components of the program that contain multithreadable program constructs or explicitly multithreaded program constructs is by source-to-source program preprocessing.

63. (Withdrawn) A method as in claim 61, wherein the result of the source-to-source program preprocessing is a program component that incorporates thread library calls representing to the transformed multithreadable program constructs or explicitly multithreaded program constructs.

64. (Withdrawn) A method as in claim 63, wherein the thread library is a thread library designed in part or whole for the purpose of representing the transformed multithreadable program constructs or explicitly multithreaded program constructs.

65. (Withdrawn) A method as in claim 63, wherein the thread library is an existing thread library or a thread library designed for another purpose.

66. (Withdrawn) A method as in claim 61, wherein the result of the source-to-source program preprocessing is a program component that incorporates standard multithreaded program constructs supported by the existing programming system.

67. (Withdrawn) A method as in claim 59, further comprising renaming the standard compiler-linker and the standard compiler-linker name is used for a program component transformation tool that subsequently invokes the renamed standard compiler-linker.

68. (Withdrawn) A method as in claim 59, wherein the operating system is Linux or another variant of the Unix operating system and the existing program development environment is the GNU C or C++ compiler or any other C or C++ compiler that operates under the given variant of the Unix operating system.

69. (Withdrawn) A method as in claim 59, wherein the existing programming language is a variant of the Java programming language and the thread library is the standard Java thread library.

70. (Withdrawn) A method of operating a program operation, comprising:

defining a block of code which can be executed either sequentially or substantially simultaneously via separate loci of execution;

running the program during a first mode in said sequential mode, and running the program during a second mode in said substantially simultaneous mode.

71. (Withdrawn) A method as in claim 70 wherein said definition is an equivalence annotation.

72. (Withdrawn) A method as in claim 71 wherein said equivalence annotation is a pragma.

73. (Withdrawn) A method as in claim 70 wherein, during said sequential execution, variables are shared.

74. (Withdrawn) A method as in claim 73 wherein said shared variables can be checked, and operation of check does not suspend operations of the program.

75. (Withdrawn) A method as in claim 70 wherein during said substantially simultaneous operations, variables are shared.

76. (Withdrawn) A method as in claim 70 further comprising debugging a program in said sequential mode and

running a debugged program in said substantially simultaneous mode.

77. (Withdrawn) An object for synchronizing among multiple threads, comprising:

a special object constrained to have (1) an integer attribute value, (2) an increment function, but no decrement function, and (3) check function that suspends a calling thread.

78. (Withdrawn) A method as in claim 77 wherein said check function suspends a calling thread for a specified time.

79. (Withdrawn) An object as in claim 78 wherein said object includes a list of thread suspension queues.

80. (Withdrawn) An object as in claim 77 further comprising a reset function.

81. (Withdrawn) An object as in claim 77 wherein said object is a counter.

82. (Withdrawn) An object as in claim 77 wherein said object is a flag having only first and second values.

83. (Withdrawn) A method of integrating a thread management system with an existing program development system, comprising:

first, running a pre-program development system that looks for special annotations which indicate multithreaded and multithreadable block of code;

using said special layer as an initial linker; and

then, passing the already linked program to the standard program development system.

84. (Withdrawn) A method as in claim 83 wherein said program is a C programming language.